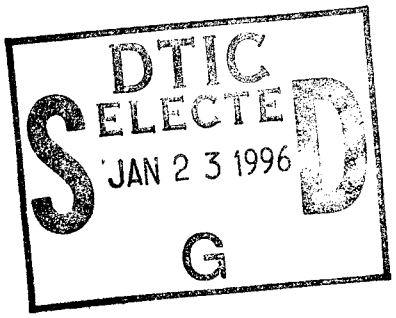
RL-TR-95-104
Final Technical Report
June 1995

REAL-TIME EMBEDDED HIGH PERFORMANCE COMPUTING: COMMUNICATIONS SCHEDULING

The MITRE Corporation



Sponsored by
Advanced Research Projects Agency

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

Rome Laboratory
Air Force Materiel Command
Griffiss Air Force Base, New York

This report has been reviewed by the Rome Laboratory Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RL-TR-95-104 has been reviewed and is approved for publication.

APPROVED: Paul Sierak

PAUL SIERAK
Project Engineer

FOR THE COMMANDER: John A. Graniero

JOHN A. GRANIERO
Chief Scientist
Command, Control & Communications Directorate

If your address has changed or if you wish to be removed from the Rome Laboratory mailing list, or if the addressee is no longer employed by your organization, please notify RL (C3CB) Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REAL-TIME EMBEDDED HIGH PERFORMANCE COMPUTING: COMMUNICATIONS SCHEDULING

Richard A. Games
Arkady Kanevsky
Peter C. Krupp
Leonard G. Monk

Contractor: The MITRE Corporation
Contract Number: F19628-94-C-0001
Effective Date of Contract: 01 October 93
Contract Expiration Date: 01 October 94
Short Title of Work: Real-Time Embedded High Performance Computing: Communication Scheduling
Period of Work Covered: Oct 93 - Oct 94

Principal Investigator: Richard A. Games
Phone: (617) 271-8081
RL Project Engineer: Paul Sierak
Phone: (315) 330-4065

This research was supported by the Advanced Research Projects Agency of the Department of Defense and was monitored by Paul Sierak, RL (C3CB), 525 Brooks Rd, Griffiss AFB NY 13441-4505.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	June 1995	Final Oct 93 - Oct 94

4. TITLE AND SUBTITLE
REAL-TIME EMBEDDED HIGH PERFORMANCE COMPUTING: COMMUNICATIONS SCHEDULING

5. FUNDING NUMBERS
C - F19628-94-C-0001
PE - 62702F
PR - MOIE
TA - 74
WU - 11

6. AUTHOR(S)
Richard A. Games, Arkady Kanevsky, Peter C. Krupp, and Leonard G. Monk

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
The MITRE Corporation
Mail Stop E025
202 Burlington Rd
Bedford MA 01730

8. PERFORMING ORGANIZATION REPORT NUMBER
MTR 94B0000146

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington VA 22203-1714

Rome Laboratory (C3CB)
525 Brooks Rd
Griffiss AFB NY 13441-4505

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
RL-TR-95-104

11. SUPPLEMENTARY NOTES
Rome Laboratory Project Engineer: Paul Sierak/C3CB/(315) 330-4065

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

The final report frames the communication scheduling problems associated with realistic real-time embedded applications on scalable massively parallel processors. These applications require high sustained processing rates, high sustained message passing rates, real-time services at the processing modes, and the most significant risk area of real-time intermode communication services.

14. SUBJECT TERMS
Parallel processing, Real-time, Embedded, Communication scheduling

15. NUMBER OF PAGES
64

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

ABSTRACT

The Government's High Performance Computing and Communications initiative has reached the stage where it is now important to demonstrate realistic applications on scalable massively parallel processors (MPPs). Real-time embedded applications on MPPs will require high sustained processing rates, high sustained message passing rates, real-time services at the processing nodes, and real-time internode communication services. Based on our current practical and theoretical investigations we conclude that the last of these requirements currently poses the most significant risk in the transitioning of programmable MPP technology to real-time embedded applications. MITRE is currently developing a scalable solution to the real-time communications problem for MPPs. This paper frames the communications scheduling problem, describes MITRE's approach and progress so far, and assesses priorities for future work in this area.

PREFACE

This is one of three MITRE Technical Reports documenting work performed during Fiscal Year 1994 on MITRE Project 74110, Real-Time Embedded High Performance Computing. The complementary reports are

- *Real-Time Embedded High Performance Computing: Application Benchmarks*, MTR 94B145, by Curtis P. Brown, Mark I. Flanzbaum, Richard A. Games, and John D. Ramsdell, and

- *Parallel Implementation of the Planar Subarray Processing Algorithm*, MTR 94B114, by Richard A. Games and Daniel S. Pyrik.

In addition, this document reports on work during the same fiscal year on MITRE Project 820W, Real-Time Communications Scheduling, which was initiated in the summer of 1994 to concentrate on the subjects covered here.

ACKNOWLEDGMENTS

This work was supported in part by the U.S. Air Force Electronic Systems Center and performed under MITRE MOIE Project 74110 of contract F19628-94-C-0001, managed by Rome Laboratory/C3CB.

This work was supported in part by the Advanced Research Projects Agency and performed under MITRE Project 820W of contract DAAB07-94-C-H601, managed by the Naval Command, Control and Ocean Surveillance Center, RDT&E Division (NRaD).

This work was supported in part by a grant of HPC time from the DoD HPC Major Shared Resource Center, Wright Patterson Air Force Base.

We wish to thank the Honeywell Corporation, Space Systems, Clearwater Florida for use of their Paragon.

TABLE OF CONTENTS

SECTION 1

INTRODUCTION

The process of specifying, designing, manufacturing, and supporting complex digital systems, particularly real-time embedded signal and data processors, must change greatly if future system requirements are to be met within shrinking military budgets. High performance computing can play a significant role in the rapid production of military systems with reduced life-cycle costs. If the computational requirements of current and future real-time embedded systems can be satisfied by emerging programmable massively parallel processors (MPPs), then costly application-specific processors and disparate data processors can be replaced by a single homogeneous, scalable, programmable computing platform that is designed to be able to take advantage of the roughly foreseeable progression of commercial technology.

There have been significant advances in hardware technology that make such a high performance computing solution possible: processing, I/O, and memory capabilities are continuing to improve. The packaging problems associated with embedded applications are currently being addressed by a variety of ARPA and industry research and development programs. In particular, the ARPA-Honeywell Embedded Touchstone program is producing an embeddable high performance computer that incorporates commercial microprocessors running the same system and application software as its commercial counterpart, the Intel Paragon.

However, software technology for high performance computing has been lagging. There is a general perception in the user community, especially in industry, that there is a need for improved programming environments and tools for distributed memory massively parallel processors. However, this is not our concern here as there are potentially more serious show stoppers in the case of embedded applications. These applications have requirements not found in large-scale scientific computing such as the need for real-time processing, multi-level security, and fault tolerance. In this paper we focus on applications that have hard real-time requirements. Results must be computed by strict deadlines or they lose their value, potentially causing catastrophic system failure. Specialized system software is needed to provide such hard real-time guarantees.

1.1 THE COMMUNICATIONS SCHEDULING GAP

Four noteworthy prerequisites for a successful real-time embedded MPP implementation are

1. high sustained processing rates,
2. high sustained message passing rates,
3. real-time services at the processing nodes, and
4. real-time internode communication services.

Here they are ordered according to our assessment of the magnitude of associated risk, starting with the least risky.

High sustained processing and message passing rates will be needed to minimize the size, weight, and power requirements of the embedded processor. At those nodes where processing is intense, and on heavily used data channels, we believe these rates need to be and can be considerably higher than the small percentages that are sometimes accepted. We have in mind the goal of achieving at least 50% of advertised peak rates as a rule of thumb, although this goal is complicated by the probability that the mapping of the algorithm will require periods of quiescence.

We expect that optimized library routines and the emerging "lightweight" message passing techniques will be able to provide the necessary efficiencies for the first two requirements above (Brown, 1994). This is especially true if the software developer is prepared to invest the effort that is typically involved with present embedded processors.

But just obtaining efficiency and speed does not guarantee real-time performance, which has more to do with predictability. This realization applies both to processing and to communication. On the processing side it has already motivated the development of real-time operating systems. For example, commercial real-time operating systems, such as VxWorks, PSoS, LynxOS, have been ported to many of today's microprocessors and incorporate the hard real-time scheduling theory developed throughout the 1980s. These operating systems are well understood by the embedded computing community and hopefully can be streamlined to provide the basic services required in a pragmatically selected near-term demonstration. More general solutions are under development. ARPA/CSTO is funding

Honeywell and the Open Software Foundation to provide real-time extensions to Mach, which then can be incorporated into the operating system of the Embedded Touchstone. We feel overall that the risk associated with the third prerequisite, obtaining real-time performance on a single processing node, is also manageable, although there will be added difficulties associated with multiprocessor nodes.

Internode communication lies at the heart of the scalable MPP concept. Even in the simplest paradigm, which imagines dividing a large problem into N parts to be solved on N processing nodes independently, the input data must be communicated to the various processors, and their results must usually be accumulated for at least rudimentary further processing. An important motivating phenomenon comes into play when there is a second stage of processing that is also parceled out to N nodes, such that each of the second stage subproblems requires some data from each of the first stage partial answers. At least $(N-1)^2$ unidirectional data transfers are necessary in this case, which corresponds to a "corner turning" common in signal processing. But each transfer is probably of smaller size than in the first stage distribution. What is clear is that an evaluation of the overall communication problem needs to consider offsetting factors, including the connection topology and other aspects of the hardware architecture and underlying software. As N gets large, the scaling of the processing is conceptually simpler than the scaling of the communication.

An overall solution to the problem of building real-time scalable MPP applications must have a component that provides guaranteed real-time internode communication performance. This component may be conceptual and may not require elaborate extra software. It is possible that for many individual cases a careful analysis of the applications software, together with a supporting theory of very limited scope, will suffice. Convincing scalable methods of analysis sufficient even for individual applications are currently wanting. In the longer term, a more general and easier to apply real-time communications scheduling facility is highly desirable, just as it is desirable to go beyond hand-crafted solutions to the scheduling problem for complex systems on a single processor.

No real-time communications scheduling component is included in existing real-time operating systems. It is also important to note that techniques developed for network applications where only statistical or other relatively "soft" real-time guarantees are required, although suggestive, may not be adequate for hard real-time applications. This limitation is significant even though it may not always be possible to make an absolutely sharp distinction between hard and soft real-time requirements.

It is the real-time communications scheduling requirement that currently poses the most significant risk standing in the way of transitioning programmable MPP technology to real-time embedded applications.

1.2 COMMUNICATIONS SCHEDULING

We are currently developing an approach to solving the communication scheduling problem for scalable MPPs. In order to get a concrete understanding of realistic requirements and to achieve early demonstrable progress, we are initially emphasizing the class of signal processing applications. But we are also being extremely careful not to tie ourselves down by this initial selection, and we are keeping more general problems in mind in all phases of the work. It is an important research question just how generally applicable a communications scheduling facility can be, just as it is with hardware architectures, operating systems, etc.

The hard real-time computing research community has begun extending their work to distributed implementations, but the work to date has focused either on packet switched wide-area networks, such as the Internet, or on local-area networks, such as FDDI. Assumptions of intelligent routers and relatively long latencies interfere with the applicability of much of this work to MPP backplanes, but some aspects, such as the protocols suggested for local-area networks, may well play a role in the solution.

What will an MPP communications scheduling solution look like? As in the case of process scheduling for a sequential real-time system, there will be two parts:

1. algorithms and protocols, etc. that can be (and are) implemented on various machines to provide adequate real-time communications, and

2. an overall analytical framework that enables us to state guarantees and easily build, modify, and compare systems.

We do not want to assume that the need for communications scheduling will be high on the list of considerations in the design of computer architectures or operating systems, although in the long run some influence might be possible. The pragmatic strategy characteristic of our approach requires us, at one end, to define performance parameters for the combinations of hardware and support software under consideration that are really measurable or predictable with a high degree of confidence and that are informative enough to feed into the rest of the

theory. Thus the simplistic concept of message passing rates is inadequate at least because it does not tell us much about how the data communications network, including the relevant software at the nodes, will behave under more complex circumstances (see also Section 3.4).

At the other end we must define requirements parameters that adequately express the communications needs of target applications. These would be analogous to the parameters associated with sets of periodic tasks that are used in the established theory of single resource scheduling. Scheduling algorithms and protocols, mechanisms for implementing them, and appropriate mathematical analysis must also be developed to justify a translation of applications requirements into performance parameters for the hardware and support software that we can guarantee are sufficient.

An attractive approach that we are exploring is to divide up the MPP backplane (or, better, the use of the backplane) either spatially or temporally or both to reduce the communication scheduling problem to smaller resource contention problems that can be handled by simple adaptations of known techniques. This reduction may be in more than one stage. A key element of this "network subdivision" approach is its explicit willingness to pay a reasonable price in efficiency to achieve the predictability that such a reduction provides. The associated analytical frameworks for each of the pieces must be integrated to obtain an analytical framework for the entire backplane. This integration may be facilitated by our suggested revision of demand notions. The network subdivision idea will be discussed in more detail in Section 3.

1.3 REPORT ORGANIZATION

In Section 2 we describe in more detail the issues and the current state of the art for the four components of a real-time embedded MPP implementation. In particular, we frame the MPP communications scheduling problem in Section 2.4. In Section 3 we describe our approach to solving this problem. A conclusion summarizes the paper and assesses how far we have gotten and what needs most to be done by ourselves or others.

SECTION 2

REAL-TIME EMBEDDED MPP

In the next four sections we review the issues involved and discuss the current state of the art for each of the four prerequisites for a successful real-time embedded MPP (massively parallel processing) implementation enumerated in the introduction: high sustained processing rates, high sustained message passing rates, real-time services at the processing nodes, and real-time internode communication services.

2.1 HIGH SUSTAINED PROCESSING RATES

To meet the size, weight, and power constraints of current and future embedded applications, the programmable processor will have to deliver a large sustained percentage of peak processing rates. This will be especially true in the case of signal processing, where the issue will be whether programmable MPPs can achieve the necessary processing densities to replace traditional application-specific processors.

Many signal processing algorithms spend a large part of their effort repeatedly executing one or a few basic operations called processing kernels. Fortunately, research has demonstrated that it is possible to attain high processing efficiency on programmable processors for most common signal processing kernels. For example, (Linderman, 1992) has described benchmark results for QR (Quadratic Residue) Factorization and space-time processing in excess of 50% utilization through the use of optimized library functions.

We have had similar experience in our recent work implementing the FoldFFT processing kernel of the Planar Subarray Processing synthetic aperture radar algorithm, see (Games, 1994). The FoldFFT function consists of a sequence of processing steps, including vector operations, array indexing, and an FFT (Fast Fourier Transform), that makes it useful for comparing, or benchmarking, the performance available from current processor technology.

We compared the single processor performance of the SKY Computer SKYbolt and Paragon GP processing node for the FoldFFT function. Three different programming approaches were considered: plain C, C with optimized library calls,

and pure assembly language. Each successive approach involves more programmer effort and is less easily ported to other platforms, with the more involved approaches providing increased performance. In this case optimizing for the block length of the FFT was a key consideration. For anticipated size FoldFFT calls the assembly language versions achieved more than 50% of advertised processor utilization.

As commercial MPP vendors increase the number of microprocessors at each processing node, attaining the required processing efficiencies will become more of an issue. For example, Intel will soon be selling Paragons in which the GP processing nodes, which contain two i860XP microprocessors one of which nominally was dedicated to message passing, are replaced by MP processing nodes containing three i860XP microprocessors. But these multiprocessor node architectures are usually based on a shared memory computational model for which there is considerable experience and tools available for obtaining high sustained processing rates.

We mention in passing the current interest in application-specific processing nodes to improve the processing density that these scalable MPPs can deliver. Although this is a deviation from the commercial general-purpose programmable stricture, it can be viewed as the next logical step beyond the use of optimized "library" routines for applications where size, weight, and power constraints are an overriding concern.

In general it appears that massively parallel platforms can provide adequate processing performance at individual nodes for partitioned algorithms that use only a few optimizable processing kernels. Application developers may have to customize generally available software to achieve acceptable efficiencies.

Obtaining high sustained processing rates at individual nodes is clearly not sufficient, as the MPP concept requires that the processing be distributed amongst the processing nodes. Historically, the need to coordinate between processing nodes has resulted in a dramatic drop of efficiency over what could have been attained on a single node. The mapping of the algorithm to the MPP should attempt to minimize the negative impact of such coordination. Again there are time-tested techniques, such as pipelining with concurrent communication and processing, that can mitigate such effects for many of the linear algebra algorithms that may benefit most from an MPP implementation. These algorithms also usually have simple nearest neighbor communication requirements, which can also minimize the negative effects of message passing. In particular for the Paragon, many of these algorithms can be mapped to a mesh with the property that messages do

not have to turn corners, which simplifies a number of issues, as we shall see. We have obtained promising results for critical cases of pipelined algorithms on the Paragon (Brown, 1994). The pipeline successfully hides the communications overhead and maintains high processor efficiencies. Note that this amounts to positive practical evidence for our assignment of low risk to the combination of both of the first two requirements listed above.

A key consideration in determining the granularity involved in this mapping problem is how fast data can be communicated between processing nodes, a subject that we now consider.

2.2 HIGH SUSTAINED MESSAGE PASSING RATES

Programs that use explicit message passing must surmount several barriers to be able to exploit the full bandwidth of the underlying message passing hardware. For example, care must be taken to minimize the number of times a message is copied. Furthermore, high performance programs must avoid message passing protocols which depend upon large numbers of messages to synchronize data transfers, because these messages incur a cost.

The copying restriction is particularly important. For example on the Paragon, the cost of copying messages was identified as a limiting factor by the authors of the Performance oriented, User-managed Messaging Architecture (PUMA) operating system (Wheat, 1993). They report achieving inter-node communication rates in excess of 160 MB/s, but they were only able to attain memory copy rates of 70 MB/s on the same machine. When communications require a memory copy, the throughput drops to about 55 MB/s. But PUMA provides an abstraction, called portals, which give programs on one node the ability to write directly into a prespecified portion of the address space of a program running on another node. Using portals, programmers can write applications in which messages are never buffered in the kernel, which greatly reduces the overhead associated with passing messages. In a similar vein, Intel has recently proposed the Virtual Channel Facility that gives the illusion that the network implements dedicated direct connections between processing nodes.

Well-behaved applications often do not require the level of buffering provided by general purpose message passing primitives. For example, there are some applications that send messages only of one fixed length, and generate a message passing pattern which guarantees that no node will have more than one unprocessed mes-

sage at any time. Programs that exhibit this pattern of message passing can be implemented with a minimum of message copying and a message passing protocol that incurs no overhead.

Many massively parallel processors provide vendor specific facilities on which a programmer can implement message passing primitives tuned to a particular application. For example, CM-5 programmers can use both active messages (von Eicken, 1992) and virtual channels to reduce the overhead associated with general purpose message passing.

The Message Passing Interface Forum has designed machine independent primitives that give programmers more control. For example, programmers can declare that point to point communication will take place in something called "ready mode." The use of this mode places additional requirements on programmers, but implementations are free to make assumptions which allow improved and predictable performance.

Unpredictability of performance of message passing primitives is also a worry. Our experience with the NX message passing primitives on the Paragon suggests that their performance varies greatly for subtle reasons. The primitives usually provide several modes of operation with different amounts of message copying. The dynamic state of the computation usually determines the message passing mode. Fast modes can be employed when the system can detect well-behaved message passing patterns. The slow modes of data transfer are required to support the illusion that the size of every message queue is "large enough." To achieve maximum performance one must isolate and control all relevant parameters, which are not necessarily well documented or stable under software updates.

Nevertheless, the situation for message passing rates is analogous to that for processing rates. In our use over the course of a year of the NX message passing on the Paragon we improved from 24 MB/sec to 86 MB/sec. See (Brown, 1994) for details of our relevant experience. It seems likely that applications developers can gradually improve and stabilize message passing performance on simple patterns by getting better or more suitable underlying software and by learning the ins and outs of the message passing primitives they do use.

2.3 REAL-TIME SERVICES AT THE PROCESSING NODES

Current research in real-time systems, especially in the area of operating systems and scheduling, indicates that the management of large numbers of concurrent processes in a manner that predictably meets timing constraints is a difficult problem. The scheduling and resource allocation must be done in an integrated fashion across the entire system, including I/O scheduling. I/O scheduling is especially critical to embedded real-time systems. The solution to this problem will require the development of new operating system technology. However, recent developments in the area of real-time operating systems and communications should make this a feasible objective. Predictable hardware architecture will also play an important role.

Traditional operating systems evolved to support time-sharing applications. The design objectives were to support a large number of users on a single CPU and to provide each user with essentially a virtual processor. This virtual processor (a process) is the functional equivalent of a dedicated machine. But since it is implemented, of course, by multiplexing the resources of a single machine among many users, the timing performance generally degrades as the number of users increases. The operating system, through scheduling of user processes, attempts a fair allocation of resources in which the users share equally in the loss of performance. UNIX is the quintessential example of this type of operating system.

In contrast, a real-time operating system must explicitly limit the degradation of the timing performance of all processes as the number of processes increases. The user must be able to specify which processes are critical and must be able to guarantee that critical processes meet their timing constraints for specified workloads. If these workloads are exceeded, it would be desirable to specify the criticality of processes so that the performance of the system degrades in a graceful, predictable manner. Current research has shown that this is a difficult problem to solve in general, even on a single processor system.

Many real-time systems (defense and space systems) have been successfully developed in the past despite the inadequacies of traditional operating systems. The problem was solved, at great cost, with custom-built real-time executives and hardware support. High performance and predictability were the design objectives. The real-time executives, while meeting the real-time constraints, were so tightly tied to the specific application that they could not easily be reused and

changes in requirements often led to expensive and risky changes in the real-time executive.

The real-time executives were often cyclic executives in which every event (use of the CPU, sensor, database) was explicitly scheduled. Until recently this meant these custom cyclic executives were often the most effective solution. Even beyond the economic issues, in a rapidly changing world, it is not always possible to modify the real-time executive in a timely manner to meet new system requirements because the required changes to a real-time executive may require a global restructuring of the executive. The provision of increased computational power by a MPP, if not adequately supported by a real-time operating system, could compound the development problems encountered in the past.

Many experts feel that the first potentially practical advance beyond custom real-time executives to general purpose real-time operating systems was the development of rate-monotonic scheduling (RMS) (Liu, 1973). The initial formulation of RMS made so many restrictive assumptions that it could not be applied to an actual application. Research has since dramatically extended the initially limited applicability of RMS and even shown how it can be used for some distributed systems and communications scheduling (Sha, 1993) (see also discussion below on communications scheduling). Other approaches have emerged over the past decade. For example, the window protocols of (Zhao, 1990), have been incorporated in various research real-time operating systems. Commercial real-time operating systems incorporating RMS are available that address the problem of scheduling tasks on a single processor, but not the more general problem of distributed/parallel real-time systems nor the problem of fault-tolerance.

Commercial real-time operating systems for the most part evolved from either UNIX and Mach or from small real-time proprietary executives. Both UNIX and Mach present the problems described earlier of an operating system that evolved from time-sharing applications. Modifications to UNIX have included the incorporation of support for RMS scheduling and priority inheritance protocols for some shared resources. (LynxOS is a good example.) They address the problems of predictable interrupt handling times, context switching times, preemptable kernels, and user assignable priorities. They also have the advantage of the development tools that are generally available in a UNIX environment and support emerging real-time standards such as POSIX 1003.4.

Support for TCP/IP provides a convenient means for these commercial systems to interoperate with existing non-real-time systems. This interoperability is limited

since the TCP/IP support does not support meeting real-time constraints, for example. Because of the UNIX heritage, other factors affecting predictable real-time performance are not addressed. There is no support for communication scheduling or real-time distributed processing. Unpredictability due to virtual memory, disk and network I/O, and X-windows servers can induce severe priority inversions (where a high priority task must wait for a low priority task).

There are several research efforts underway in the area of real-time operating systems: Alpha, MARS, Spring, MARUTI, ARTS, CHAOS, HARTOS, DARK, RT/Mach (Ramamritham, 1994). They all attempt to address the issue of an integrated approach to scheduling. None of them explicitly address the issue of real-time operating system support for an MPP, although they all provide some support for distributed real-time applications.

A distributed real-time operating system should provide support for (1) basic guarantees for real-time constraints, (2) fault tolerance, (3) distribution across a multi-processor heterogeneous system, and (4) integrated time-constrained resource allocation. The distributed nature of the system would require that time-constraints are enforced for collections of tasks not just individual tasks. The timing constraints of all operating system and infrastructure primitives and their interactions would need to be known and be predictable. The operating system and infrastructure should also support scalability.

A real-time operating system for an MPP would address similar issues but the homogeneous nature of the processors and regular topology of the communication network should simplify the problem. Operating systems and languages for MPPs have traditionally been oriented towards the support of scientific computing in which the principal objective is to attain as much of the potential speedup offered by a parallel architecture as possible. While high throughput is desirable for real-time applications on an MPP, timing predictability must not be sacrificed. None of the current MPP operating systems, communication libraries, or languages take explicit account of real-time predictability. The need for predictable communication in an MPP complicates the real-time operating system problem.

2.4 COMMUNICATIONS SCHEDULING

In order to guarantee the hard real-time performance of the overall system it is clearly necessary to ensure timely flow of data among the computing tasks. This is relatively easy for tasks on the same processor, but may be difficult when message

passing on some sort of network must be used. It is particularly difficult when an attempt is made to maximize the usage of processing resources by pipelining and when the data communications network is complex.

Real-time (especially hard real-time) applications have not been uppermost in the minds of the developers of multiprocessors, and certainly much of the responsibility for this component must inevitably be borne by the developers, not of multiprocessors, but of operating systems and applications software. In fact, there is so far not a lot of experience to draw on to relate the simplistic performance measures that are available for new massively parallel machines to effective performance in actual realistic applications, even other than hard real-time applications. As discussed below, existing scheduling technology may be farther from providing an adequate solution than one might expect.

The scope and depth of the problem of how to provide adequate communications scheduling for real-time embedded massively parallel processing as we see it can be seen better by discussing four different aspects.

1. The distribution of the communications resource: We want to achieve effective simultaneous use of physically separated parts of a communications network as much as possible. It would largely defeat the purpose of the hardware architecture if we were reduced, for reasons of predictability, to using the sophisticated data communications network of a supercomputer like the Paragon or CM-5 as if it were a multi-access bus.

2. The distribution of knowledge of communications needs: We cannot assume a centralized repository of current communications requests. Some communication needs will be known ahead of time, for example, from the preliminary algorithm partitioning and mapping, while other communication requirements can only be known at run time, because of data dependencies in the computation. Can useful bounds be determined in the latter case?

3. The weakness of available control mechanisms: The backplane mesh of the Paragon, for instance, provides fixed message routing algorithms that do not allow for any sophistication when contention arises. Indirect measures by entities off the mesh itself are probably necessary. They may take a while to appreciate what is going on on the network and also to affect it. Without a basic architectural modification, our mechanisms for implementing whatever distributed scheduling algorithm we may decide to use must depend mostly on the operating system and protocols for cooperation observed by the applications processes themselves.

4. The relationship of communications scheduling to other components of the overall problem mentioned above: On one side is the high performance message passing. Benchmarks, requirements, and guarantees must be designed that reflect the kind of performance that will be necessary for a real embedded system, not just for high volume, low latency node-to-node channels. We will need to know how the communications system performs under a multidimensional variety of loads.

On the other side are the operating system, applications processes, and the more or less periodic message passing requirements that will be generated. Simply guaranteeing that certain messages will be ready before a given time may not be sufficient, because having them ready too early could slow down the transmission of other messages. Thus the importance of lack of "jitter" is amplified in this context. Calculations of required buffer size may also be difficult.

To this list might well be added various other problems, such as possible disparities among processors' local clocks, and the possible necessity for distributed input and/or output.

To emphasize the inadequacy of separately providing high performance message passing, high performance processing at the nodes, and a high quality real-time operating system at the nodes, without addressing the communications scheduling component in the integrated way we advocate, consider the following example for what might happen on a machine like the Paragon. Suppose that ten nodes arranged in a horizontal row (N_1, ..., N_{10} from left to right as shown in Figure 1) are all generating messages at a high rate and sending them to N_0, which lies even to the left of N_1, the leftmost of these senders. We assume that the message passing software is good enough so that each node can generate messages sufficiently fast to use up the capacity of a horizontal link.

A reasonably designed backplane that cannot take into account message origin or priority might use a fair merge at each horizontal node (this is the case with the Paragon). In this case, data will flow from N_{10} to N_9 at a rate of, say, C bytes per second. Similarly, C bytes per second will flow from N_9 to N_8, but half of this flow will be from N_9 and half originally from N_{10}. Unfortunately, the extension of this fair merge policy further down the line becomes disastrous, because the fraction of total flow on each horizontal link that originates from N_{10} decreases by half at each step, so less than one byte per thousand flowing into N_0 originates from N_{10}. We call this phenomenon the failure of fairness.

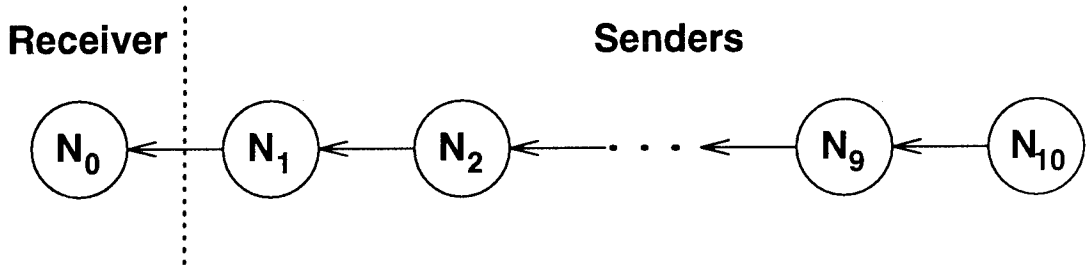


Figure 1. Communication Example

It seems likely that for certain pipelined cases this drastic reduction in the bandwidth available to N_0 will be transient and will not be a serious problem.

With lower utilization the performance penalty for successive fair merges of additions to a data stream will be less extreme, but still considerable. We believe that we have observed its effect on the Paragon in our benchmarking experiments, but only when using multiple receivers, so that the slow removal of messages from the backplane by N_0 doesn't dominate other costs (Brown, 1994).

A related degradation in predictability is especially relevant to hard real-time applications in which typical messages comprise only one or a few packets. A single packet X traveling from N_{10} to N_0 will be delayed at an intermediate node if that node just happens to be putting one of its packets into the stream slightly before the arrival of the head of X, or if it puts on a packet at essentially the same time as the arrival of the head of X and X loses a (fair or unfair) arbitration. With the Paragon system, X can get no credit down the line for having been delayed earlier. Nor can X be given consideration for being part of a large or important message. Even with fair arbitration the possible variation in the arrival time of X is large.

Where are we to look for a solution to the communications scheduling problem? As in the case of process scheduling for a sequential real-time system, we are looking for two things:

1. algorithms and protocols, etc. that can be (and are) implemented on various machines to provide adequate real-time communications, and

2. an overall analytical framework that enables us to state guarantees and easily build, modify, and compare systems.

A natural suggestion would be to treat message passing as much as possible like any other task and try to transfer successful scheduling technology more or less intact. The distributed nature of communications and the lack of control mechanisms on the backplane are significant obstacles to such an approach, even if one is not ambitious about performance. An efficient solution is, of course, even more difficult.

It does not seem to be sufficient for our purposes to appeal to the general theory of multiple resource scheduling. The informal consensus is that this theory does not at the present time provide very useful techniques, and it seems that the special considerations of message passing are quite significant in differentiating problems of most concern to us from the general cases. These include the fact that one must worry about finding a mechanism to enforce whatever schedule one might decide to use, and that there might not be centralized knowledge of all communications needs unless the computation is absolutely without deviation from a known periodic dataflow pattern. Also, in our context there would be very many different resources to consider, which would make it unfeasible to apply some fine-grained styles of analysis.

Recent papers (Aras, 1994; Malcolm, 1994) survey and compare current approaches to the problem for a variety of networks and applications, but not including the context of interest to us. We can hope to learn much from this highly developed technology, especially for "subscheduling" localized parts of the network for short time intervals. But the difference of context is surprisingly significant. For instance, their context usually implies that the intermediate nodes that pass along data are capable of intelligent decisions. Also, most of the data traffic we envision will have hard deadlines (and the first of these papers assumes the opposite), so certain efficiencies that might be gained by filling up spare capacity with soft-deadline data are not realistic. As is said in (Malcolm, 1994): "The basic approach taken in scheduling real-time messages over a local area network is to emulate existing centralized scheduling." The network is used exclusively by at most one message at any given time. For these and other reasons one must be very careful when transferring general conclusions from standard theory to our context.

Development of a communications scheduling component is essential for high performance real-time embedded systems on massively parallel computers. It

should have an early impact on the other components, because it will provide greater insight into how to evaluate underlying message passing capabilities and the requirements to be imposed on real-time processing at the nodes. It may eventually have an impact on architectural issues. For instance, the questions of how much capability for algorithmic scheduling needs to be made available at the routers and what balance is needed, in a very general sense, between the communications capabilities and the brute computational capabilities of massively parallel machines.

SECTION 3

COMMUNICATIONS SCHEDULING APPROACH

Our motivation, goals, and expectations for our work on this component hopefully come across sufficiently well from the last section. Keep in mind that, as stated there, we are looking for both adequate methods of scheduling and an appropriate analytical framework. In this section we summarize the program of research and development that is emerging from our work up to this point. In the first three subsections we compare two general, somewhat dual, approaches to the solution (called the link level and network subdivision approaches) and describe more specific examples of the Network Subdivision approach. In the subsection on Packet Stream Analysis we present an analytical framework that is at an intermediate level of granularity and is compatible with the Network Subdivision approach (among others). A potentially useful variant of the ordinary notion of utilization is presented in the section on Discrete Demand. The relationship of a potential communications scheduling facility to other components of an overall solution is touched on briefly in Section 3.6.

3.1 LINK LEVEL APPROACH

One approach is "bottom up," considering individually many small components of the data communications network. We call this the link level approach. Each of these elementary components is treated as an individual communications resource. One can be more or less ambitious about how small to take the elementary components, but they should be chosen small enough so that their behavior as transmission devices can be pretty well pinned down. They should deal with only one, or at most very few, packets at one time. For example, a model of the Paragon might have one resource for each direction of each horizontal and vertical link of the backplane, two resources for each connection from the backplane to the compute nodes, and probably several resources for each of the specialized processors at the backplane intersection points.

There are two advantages to this kind of model. First, it provides a convincing mathematical framework to accumulate and coordinate our factual knowledge about the actual machine or its specifications. Second, it allows a reduction of communications scheduling problems to the same mathematical form as more traditional multiple resource scheduling problems. While such mathematical for-

malizations may have been developed with resource activities other than data transmission in mind, their purely mathematical analysis is valid for this context also, of course. Thus we can hope to draw on established theory or at least try to modify results that come close to using the right assumptions.

Without going into detail, the reduction involves looking upon the long distance transmission of a packet as the successive transmission of that packet through a series of atomic components. This is facilitated on some machines, like the Paragon, because the complete path that a packet must take from transmission to reception is determined in a fixed simple manner, although the times at which it reaches the various points of this path are not. The processing done by each resource is essentially just transmission, and the transmissions must be performed in exactly the right order. Thus one needs to add an ordering constraint to ordinary multiple resource scheduling problems.

Three points should be made about the link level approach as it relates to our current work. First, it is probably too ambitious. Because of the number and nature of the resources involved, satisfactory mathematical analysis becomes very difficult, and scheduling algorithms, bounds, etc. that one might extract without clever approximation tricks might well be unacceptably complex. This is only a partial rejection, because convincing approximation may be possible, and tractable general bounding results may be feasible.

A second point about the link level approach amplifies the first worry. The actual packet size of typical transmissions on the Paragon would probably be such that it often occupies many elementary components at once. This further complicates the mathematical form of the problem.

The last point about the link level approach is more positive. It may allow one to provide crude, but convincing bounds on the performance of parts of the network in very simple circumstances. Such bounds will probably be used at some point in the rigorous justification of the results suggested by other approaches.

3.2 NETWORK SUBDIVISION APPROACH

The network subdivision approach is somewhat dual to the link level approach. It is "top down," viewing the entire communications network as a complicated device that does many things simultaneously. Different sets of tasks that it is performing simultaneously interfere with each other to different degrees. The characteristic

idea is to divide up the network (or the use of the network) either spatially or temporally or both to reduce the problem to smaller resource contention problems that can be handled by known techniques. This reduction might be in more than one stage. There is an explicit willingness to pay a reasonable price in efficiency to achieve this reduction.

An acceptable reduction must include the integration into an overall analytical framework of a global analysis (probably to be provided by us) of the process of subdivision with a known or constructible local analysis of the smaller problems (such as GRMS might provide).

We assume the overall communications requirements for the application are expressed as a set of what we call periodic message tasks. This does not mean we assume all message passing requirements are periodic. But the periodic requirements are expected to dominate initially, and techniques have already been developed in the context of ordinary hard real-time scheduling which allow a certain amount of nonperiodic and/or soft-deadline work to be handled by a special periodic task.

Each periodic message task corresponds to an indefinite sequence of messages of fixed size that need to be sent from one fixed location to another, with a specified period, message ready time, and deadline. The periodic message tasks are to be derived from knowledge of (or assumptions about) the dataflow graph of the application, the physical location of abstract computation tasks from that graph, and the real-time processing capabilities of the nodes. Guaranteed behavior of the operating system is of course required.

Note that our communications scheduling techniques must be flexible enough to allow for some play in the times at which messages are ready to be sent, and this is built into the notion of a periodic message task. Also, we probably will need to use as part of the communication scheduling component run-time decisions that are made at the processing nodes. This may include interpreting handshake messages, following token-passing protocols, and looking at a clock to delay sending messages under some circumstances. None of these should be computationally expensive, but they do require compiling additions to the basic functionality of the application into the programs that run at the nodes, and that will affect the timing of their message requirements somewhat.

Interruptions by the operating system must also be allowed for at some point. This is a potentially serious obstacle for any approach to communications scheduling,

particularly for non-real-time operating systems. We do not yet have a comprehensive plan for a solution, but it must take into account considerable prior knowledge of the temporal distribution and extent of the interruptions. There are two obvious weapons to use: explicit allowance in schedules for large, highly regular interruptions and overriding smaller ones by appropriate choice of granularity and padding. It is not obvious that these will suffice in general — one might even reformulate our problem as the generation of minimal sufficient requirements on the operating system to allow for communications scheduling.

In our current formalism the spatiotemporal division of network usage is accomplished as follows. The set of periodic message tasks is partitioned into an arbitrary number of message type families. That is, each periodic message task is assigned to one and only one family. We stipulate an activation schedule that prescribes at which times each of the various families is active. The significance of a family being active is that some lower level scheduling system is free to send message instances from periodic message tasks of a given family when, and only when, that family is active.

The activation schedule must satisfy a noninterference axiom whose statement requires that we imagine the network is divided up into a set of elementary components as in the link level model. What is important here is that the network elements are independent in the sense that simultaneous use of any group of them does not reduce the effectiveness (i.e., the bandwidth, latency or any other relevant measure) of any element. To each family F of periodic message tasks there corresponds a subset $N(F)$ of the entire network that is defined to be the smallest set of network elements such that any message instance from any periodic message task of that family will use only network elements from $N(F)$ when sent. Note that the subnetworks may be oddly shaped and need not be disjoint. The noninterference axiom is then just that

(*) for every two families F and F', if F and F' are ever active at the same time, then $N(F)$ and $N(F')$ are disjoint.

Another way of putting this is that more than one family may be active at once, but those active at any one time use disjoint subnetworks. This accomplishes the factorization abstractly, and we are ready to consider how to do the scheduling on the subnetworks. We are left with several important questions:

1. how best to partition the set of periodic message tasks

2. what relative amount of time to allot to each family to be active and how to break up and interweave their active intervals as sets of times

3. how to implement whatever activity rotation we decide upon

4. how to determine and implement communications scheduling on the subnetworks when active.

3.3 SUBDIVISION PROPOSALS

We first sketch Simple Time Division Multiplexing (STDM). In this case, the periodic message tasks are partitioned very simply by putting all messages with the same sender in the same family, not looking in any more detail. Each periodic message type is assigned a load and the relative total loads of the messages of a given family determine the fraction of the total time that family is active. Actually, it may not be easy to determine an adequate way of assigning loads, but this problem has been dealt with before by others in somewhat similar contexts. We assume here also that the rest of question (2) poses no real problem.

The fact that the families are sender based has two advantages: a simple token passing protocol can be used to implement the activity rotation, and the sender can effectively enforce whatever scheduling policy it wants during its period of activity. This is a significant payoff, because we are essentially reduced to the single resource scheduling problem in a form that has a highly developed theory. Generalized Rate Monotonic Scheduling, Earliest Deadline First, and what we call Fire When Ready all are candidates. By Fire When Ready we mean that the sender just sends all messages when they are ready to be sent (during its activity period). This may sound bad, but if the backplane is very fast compared to the process of getting messages on and off the backplane, then it may be reasonable, as well as requiring the least amount of changes in the system software.

Simple Time Division Multiplexing is very similar to the suggestion in (Sha, 1993), except that this paper is concerned with a particular system involving data communications on a ring or a bus. The essential similarity lies in the key idea of using a preliminary step to reduce the problem to an interweaved set of sender based scheduling problems. The big disadvantage of STDM in our context is that it does not attempt to make use of the distributed nature of the network, it may be unacceptably wasteful.

The second proposal is specific to machines similar to the Paragon and might be called Orthogonal Mode Scheduling. It is not meant to be applicable in all cases, but only when a certain assumption about the message flow patterns is obeyed, namely that all messages follow paths that are either purely horizontal or purely vertical. This is in fact often the case for mappings involving linear algebra functions as was discussed in Section 3.1. We do not attempt any further justification of the reasonableness of this assumption here, except to point out that one way to ensure it would be to make sure that more general messages (which may turn at most one corner) are intercepted and passed on by the node at their corner, if they do turn a corner.

In Orthogonal Mode Scheduling there are two message families, horizontal and vertical. The deterministic message routing algorithm of the Paragon can easily be seen to imply that the noninterference axiom is satisfied as long as we allow only one family to be active at once. Again we appeal to some reasonable, unspecified procedure for allocating time between the horizontal and vertical modes. The idea here is that the entire machine switches from horizontal to vertical mode with a relatively low frequency by means of the available global synchronization primitives.

Within one interval when the horizontal family is active, there is potential interference only between messages sent along the same horizontal row, and similarly for the vertical family. We now can use another layer of reduction. For instance, one might treat each horizontal row as a network itself and apply the STDM sketched above. Note that considerable parallel utilization of network elements can be achieved in this manner.

Let us introduce some terminology relevant to the Paragon at this point. The overall data communications network includes the backplane proper and what we will call ramps and node gateways. The (on and off) ramps are the connections from the compute nodes to the backplane proper and are, from our point of view, really quite similar to the backplane proper. Node gateways include everything between the applications process (or wherever one wants to start an analysis) and the ramps. The exact nature of the node gateways is not fixed. In particular, node gateways include network interface and message passing software. Thus they are sophisticated entities whose behavior may be somewhat hard to control or predict. We want to make sure that every resource for which there might be competition among messages is accounted for.

We are also considering modifications of these proposals that attempt to take advantage of the fact that, at least in some circumstances, the actual transmission on the backplane is much less of a constraint than the transmission to and from the backplane. In this case we can let $SR(F)$ be the set of compute nodes which are either senders or receivers of some periodic message task in a family F, and modify the noninterference axiom (*) by weakening it to use SR instead of N, that is, to the simple condition

(**) for every two families F and F', if F and F' are ever active at the same time, then $SR(F)$ and $SR(F')$ are disjoint.

The usage of other network elements would be suitably bounded to ensure that neglecting them is justifiable.

3.4 PACKET STREAM ANALYSIS

Here we present just an introduction to an analytical framework at an intermediate level of detail that we are developing. The framework is designed to be able to provide performance guarantees for message passing schedules that use the parallel capacity of communications networks like that of the Paragon relatively efficiently. It provides a good example of the "pragmatic" strategy mentioned in Section 1.2, p. 4, because it attempts to use performance parameters of the underlying hardware and software for which credible bounds can realistically be established. It is compatible with the network subdivision approach but not to be considered only a part of that scheduling approach.

We begin with a discussion of the granularity of events mentioned in schedules — they could be fine grained (like the arrivals of particular bytes at particular routers) or relatively coarse (like the initiations and completions of send and receive messages in user processes) — and introduce the intermediate level of granularity used in Packet Stream Analysis. Next we motivate the desirability of mixing packet streams and give a motivational, "bottom up" view of the overall capacity of the network. A central theme of this section is the desirability of using, instead, a more practical "top down" characterization of network capacity, in terms of what we call "feasible message transmission modes." The idea is that the feasibility of some message transmission modes could actually be firmly established. We introduce a notion of schedulability using schedules that are based on feasible modes. An example clarifying the accumulated sequence of formal definitions is presented, an algorithm for determining if a set of requirements is

schedulable using a given set of feasible modes is given, and the overall approach of Packet Stream Analysis is summarized.

3.4.1 The Granularity of Scheduling

It is sometimes given as one of the defining characteristics of hard real-time problems that we must consider constraints on individual messages. For example, a requirement might be that a particular message must be completely received by a particular time. This emphasis on individual messages may be a slight exaggeration, as groups of messages are often what really counts, for instance, when several inputs are required for a subcomputation to begin. Nevertheless, as a good first approximation it seems that we must schedule at least down to the level of detail of individual messages. Finer levels of scheduling are really used as a device to ensure the message level requirements, and hence we are free to choose any finer level to facilitate the scheduling theory or implementation. Thus, we might choose to schedule when each packet of each message is sent and arrives.

Packet Stream Analysis depends on a cumulative analysis at a level between packets and messages. Individual packets are not scheduled. Their precise fate is left to the vagaries of intricate or obscure lower-level software, hardware, and run-time events. Attention is focused rather on the rates of transmission in packets per second and the times at which those rates are attained.

We are motivated by situations where most messages are relatively long and are sent broken up into many packets of size Δ_{data}. There may be different packet sizes for different messages, but we start by assuming that Δ_{data} is fixed. On the other hand, the optimal value for Δ_{data} is one of the parameters that one may try to determine using this kind of analysis. Even if only one channel is considered, different packet sizes result in different data transmission rates in bytes per second, because of the overhead of packetization and possibly other factors. In the case of several channels in operation at one time, the packet size has other effects too. For instance, smaller packets result in the stream model being more valid and the smoother mixing of streams, but they require regulated sending at higher frequencies.

If a given message consisting of many packets is sent at a given time, what is most significant in determining the time of complete delivery is the rate of transmission in packets per second. This assumes that the rate is reasonably constant, and that individual packet latencies, start-of-message and end-of-message processing, and other costs are relatively small. This does not mean that all of these costs can be

neglected, but they will be treated as second-order corrections to a theory based on packet rates. Similarly, if the rate of putting message packets onto the backplane is not constant, but attains several different constant values (including possibly 0) over relatively long time intervals, then the beginning times and lengths of these intervals and the rates on those intervals are the critical factors in determining how much of the message has arrived by a given time. In a less controlled situation, where one has only bounds on a possibly fluctuating packet transmission rate, bounds on the arrival time can be calculated.

The critical times at which rates change are not necessarily determined in the schedule absolutely (i.e., by particular *a priori* determined values of any clock reference), they may be influenced by run-time events in various ways. For instance, one might begin sending a message at a small rate until some other message is known to be completed, when the rate can be raised. Here we just want to emphasize that we are starting by constraining the class of schedules which might end up being successfully used. We need to worry about there being *some* way to effect an acceptable schedule, but the exact choice of mechanisms is not a concern of this analytical framework.

If the number of packets per message is low, then a limiting case of Packet Stream Analysis might still be valid, but its justifying assumptions would certainly be stretched, and the overhead of its machinery of definitions would not be worthwhile.

3.4.2 Stream Mixing

At the packet stream level, there is an attractive way of viewing the overall carrying capacity of the communications network, namely that, with certain allowances for overhead and as a function of Δ_{data}, it can simultaneously support various packet streams moving at various rates as long as the total carrying capacity of each network element (in packets per second) is not exceeded. This might allow for an enormously simplified analysis, as long as adequate (and not too drastic) allowances can be made for all relevant categories of overhead.

Initial latency, the transmission time for the first packet of a message, is one kind of overhead. The exact nature of the network interface software and message passing software must be considered at this point, because various other costs, like memory copies, might be relevant. In the case of the Paragon architecture, one kind of overhead would be an allowance for the extra difficulty that a router has when it needs to combine two incoming streams, as opposed to just passing

one stream through. It should be optimistically noted that two streams moving in opposite directions through a backplane node do not interfere with each other, at least according to the best information we have at this point.

This view of capacity suggests the intriguing possibility that we may be able to use the Paragon backplane for real-time applications more or less as it was intended to be used in general. The Paragon backplane was designed taking into account a long process of interacting theory and practice, and its designers clearly did not have in mind that applications systems would need to take care that message streams never interfere with each other. Rather, the underlying hardware and software should take care of stream mixing efficiently. All we need to do is get hard bounds on how well this is done and make sure we never ask too much of the system.

Implicit in the above discussion is the idea that under certain circumstances we might ask a node to send a stream at a regulated rate less then the maximum possible. It is unclear how good a mechanism for doing this is available. Even assuming a good real-time operating system at the nodes, this requirement has a "negative" aspect to it that is not of the same character as the requirements usually envisioned for operating systems or message passing software. That is, for most requirements, faster is better. In as much as it seems that stream sending regulation would be much more efficient if done by software at a lower level than the application, this capability may eventually be a serious suggestion for a new requirement on real-time operating systems or message passing software that is motivated by the global communications scheduling problem.

On the other hand, good stream sending regulation is probably not essential. Note that very low rates can be effected by the applications process itself, if necessary, although how *evenly* is unclear. We would like to know when stream regulation would be desirable and how stringent are the requirements on available rates, accuracy and evenness of rates, etc. But we would also like to use this analytical approach to justify schedules which depend mostly on intelligently timed "blasting," i.e., sending packets at as high a rate as possible, partly because this is, again, the natural way to use the machine. It may be possible to provide useful guarantees for blasting when combined with regulated sending at very low rates. This highly asymmetric paradigm would be reasonable when most of the message passing traffic is high rate data transmission that has been carefully planned beforehand, but some small amount of miscellaneous message traffic is necessary.

3.4.3 Feasible Message Transmission Modes

There are several difficulties in determining the capacity of the network from the capacities of its elements and appropriate overheads that are related to the objections made above to the Link Level approach (page 20). How is one to isolate and definitively measure or credibly predict the capacities of each of the network elements involved in the transmission of a message? How are we to combine the overheads associated with the elements of even a single channel into one overall channel rate? When two streams are combined, how does the associated overhead affect them individually? This last difficulty, with related questions about combining streams, is probably the most interesting and compelling.

To see that some naive ways of combining overheads will not work, consider the following example. If there are several streams merging into one, then the failure of fairness example becomes relevant. Either the analysis of overheads is subtle enough to take into account the precise order of the senders, or the overheads are taken so high that backplane carrying capacity is not approached, and the phenomenon of failure of fairness is not manifest.

The notion of feasible message transmission modes is suggested to sidestep some of these difficulties. Instead of talking about the capacity of individual network elements, or defining the capacity of the overall network directly, we define what is essentially the capacity of a way of simultaneously using a set of (abstract) channels.

Consider the patterns of simultaneous message streams on the Paragon backplane depicted in Figure 2. All of the nodes represented are on one horizontal line, and the vertical line segments represent the connections (ramps and node gateways) from the backplane to the computation nodes. This is an important special case, as observed in the discussion of orthogonal mode scheduling (page 24), and is quite complex enough to illustrate the idea of feasible modes. In Pattern 1, A is sending to B, and C is simultaneously sending in the opposite direction to D, a node between A and B. The hardware is designed so that there should be little or no interference between these two message streams, and we have not observed any. In Pattern 2, A is simultaneously sending messages to B and C, which lie in opposite directions. Depending upon the support software, this may require the application to interweave sends laboriously. In any case, we expect for Pattern 2 less resource contention on the backplane than off it, although the router node at the connection of A to the mesh is being used by both streams. In Pattern 3, A

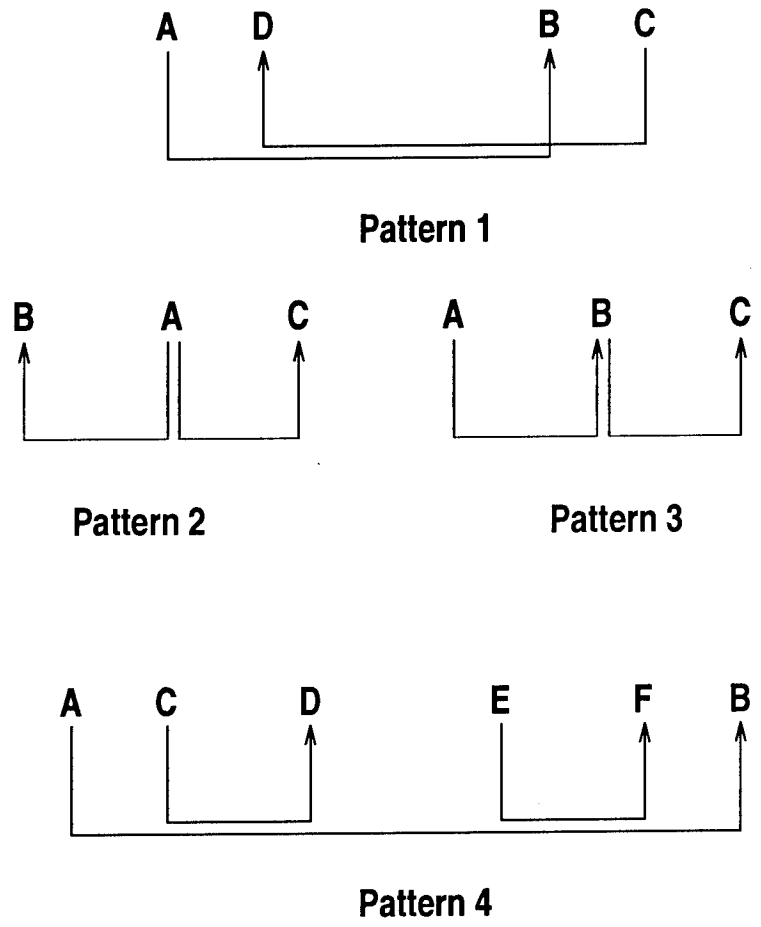


Figure 2. Some Linear Patterns on the Paragon Backplane

is sending to B, which is simultaneously sending to a node C further along, and similar comments apply.

Pattern 4 represents the kind of situation of most immediate interest. Even with a good understanding of how the hardware and software operate and measurements of achievable rates for single streams, it is difficult to predict how quickly data can be transferred in this pattern, especially assuming all senders just send as fast as possible. On the other hand, we may be able to demonstrate, through a combination of experiment and knowledge of the implementation, that certain rates (in packets per second) can be reliably established and maintained. For instance, it may be possible to use regulated sending at rates r_1 from A to B, r_2

from C to D, and r_3 from E to F. We could then think of $(r_1,\ r_2,\ r_3)$ as a lower bound on the capacity of the network relative to this pattern.

We expect that the capacity of the network for this pattern is relatively insensitive to minor distortion of the picture – we might use a vertical line, or use a different horizontal line, or move everything over to the left one unit, or even increase the distances without changing the capacity very much. It seems plausible, but not at all convincing, that keeping r_1 and r_2 the same and reducing r_3 should result in an achievable triple. It would be safer to interpolate: for instance, if $(r_1,\ r_2,\ r_3)$ and $(r_1,\ r_2,\ s_3)$ are both achievable, and t_3 is between r_3 and s_3, then $(r_1,\ r_2,\ t_3)$ is achievable.

High sending rates may only be achievable without any regulating mechanism at all ("blasting"). Especially in that case, it may be an unacceptable oversimplification to use particular rates in our description of achievable, stable message transmission modes. The rates may vary with time considerably, and the order of starting may be important. There may be hidden aspects of the situation which determine the actual rates in a given trial, so one must be very careful about assertions of what can reliably be achieved. For this reason, we use rate intervals rather than particular values for the rates in the formalization below.

First define a *channel* to be an ordered pair of nodes, a sender and a receiver. Because of the fixed routing algorithm of the Paragon, a channel corresponds to a particular path from the sender to the receiver, at least on the backplane. If one considers in detail not just the backplane, but also, as we must, what we have called the ramps and node gateways, then it may be possible for two messages from the same sender to the same receiver to take slightly different routes. One way this could happen is that there might be a buffer which is used only in certain circumstances and bypassed in others. It is thus possible that a channel may not be quite the same as a path. For our purposes we can just talk about channels and avoid worrying about paths as such. We do not differentiate among different processes at the same node in the notion of a channel, although this is a possibility for the future.

A *(message transmission) mode* is a vector of distinct channels, together with associated upper and lower bounds on the packet rates for the channels of the pattern. The lower bounds must be at least 0, which can be used as an indication of no real restriction. Any clearly unattainable upper bound can also be used as an indication of no real restriction. The channels do not have to be restricted to one horizontal or vertical line.

A message transmission mode M is *feasible* for a particular packet size Δ_{data} if simultaneous packet streams for all channels of M with packets of size Δ_{data} and packet rates consistent with the bounds of M can actually be established and sustained. It is not required in this definition that the senders be able to maintain constant rates, only rates within the given bounds. One case where this is particularly relevant occurs when we can guarantee certain of the lower bounds only by blasting. It *is* assumed that all channels sharing any network resource with any channel of M have rate bounds stipulated in M or are not in use at all. There are hidden parameters in this definition, especially the operating system and message passing software being used. Also, in as much as rate specifications are not really instantaneous but involve averaging over at least a short time interval, there is some ambiguity in the definition, but we assume this is not a problem in light of the basic assumption of many packets per message.

In order to make guarantees of schedule adequacy as discussed below, it is not necessary to know exactly which modes are feasible. It is enough to know that certain modes are feasible, and we do suppose that this is possible by a combination of experiment and analysis, at least for some relatively simple, but useful modes. The more information of this kind one has, the better analysis of communications schedulability Packet Stream Analysis provides.

3.4.4 Packet Stream Mode Schedules

A *packet stream mode schedule* (or *PSM* schedule) is a sequence of message transmission modes say $M_1, \ldots, M_k$, together with a list of times $t_1 < \ldots < t_k < t_{k+1}$. The idea is that mode M_i starts at time t_i and ends at time t_{i+1}. Overhead for switching modes must eventually be addressed, as must other sources of inefficiency, of course. A PSM schedule S, as above, is *feasible* if each of its sending modes is feasible.

We adopt a fairly standard notion of message passing requirements. In the version presented here neither schedules nor requirements are assumed to be periodic, but a more refined version should take into account periodicity. For now, an *individual message requirement* is just given by a sender, a receiver, a number of bytes, a ready time, and a deadline for arrival (which must be greater than its ready time). The *live interval* of an individual message requirement is the interval of time from its ready time to its deadline. A *simple requirement set* is a set of individual message requirements such that no two individual requirements with the same channel have overlapping live intervals.

Recall that we are assuming relatively small individual packet latencies. Given a PSM schedule and an individual message requirement it is easy to calculate, as a function of time t, a lower bound on the number of bytes of that message that will have been sent by time t, assuming that the sending modes indicated by the schedule are followed. Hence one can calculate a worst case arrival time for the entire message. We say that a PSM schedule S is *adequate* for a simple requirement set *Req* if S is feasible and, for every individual message requirement R in *Req*, the worst case arrival time thus calculated using S is at most the deadline for R.

Given a packet stream mode schedule S which is adequate for a set of requirements *Req* and faced with a system whose communications requirements are described by *Req*, there are still implementation issues to be faced. At some point one must make allowances for time necessary between modes to coordinate all senders' changes to their next determined rates and receivers. The appropriate mechanism for switching modes is not clear either. If sufficiently good global synchronization is available, then one can simply insert modest pauses to allow for all packets of the last mode to be cleared and for all senders to be assured of having reached their idea of when the switch is to occur. As an optimistic example, the relative difference between the hardware supported local clocks at any two nodes of the Paragon at any given time is guaranteed to be at most one microsecond.

One detail deserves comment here. Note that we have stipulated that the packet streams are always to be sent at rates between the currently relevant upper and lower bounds. We assume below that artificial padding can be sent (to satisfy the lower bound requirement) in case a schedule calls for a message stream to be sent before the message is really ready or after it has been completely sent. It is not out of the question that such padding might be necessary for the smooth and hence efficient operation of the message passing system. If such padding is necessary but not feasible, then more complex definitions are needed.

3.4.5 The Schedulability Problem

Given a simple requirement set *Req*, is there a PSM schedule which is adequate for *Req*? This can be taken to be the fundamental *schedulability problem* for Packet Stream Analysis. There are many, many variants (for instance, one should be able to produce an adequate schedule if there is one, and periodic versions should be considered). The schedulability problem is representative of the difficulties involved with related problems, and we concentrate on it for definiteness.

Example: Suppose we have three message requirements along the three channels depicted in Pattern 4 of Figure 2: one message from A to B of length L_{AB}, one message from C to D of length L_{CD}, and one message from E to F of length L_{EF}. All messages are ready at time 0, and all have deadline 1. Assume we know that three modes (M_1, M_2, and M_3) are feasible, where M_i has lower rate bounds of x_i, y_i, and z_i respectively for channels $A \rightarrow B$, $C \rightarrow D$, and $E \rightarrow F$, and where the corresponding upper bounds are x'_i, y'_i, and z'_i.

Is there a PSM schedule that is adequate for these requirements and uses only these modes? It should be clear that we may legitimately restrict attention to schedules that start with M_1 from time 0 to time t_1, then switch to mode M_2 for a duration of t_2, and end with mode M_3 for a duration of t_3. Such a schedule is adequate for these requirements if and only if the following seven inequalities are satisfied:

(1–3) $0 \leq t_i$, (for $i = 1, 2, 3$)

(4) $t_1 + t_2 + t_3 \leq 1$

(5) $x_1 t_1 + x_2 t_2 + x_3 t_3 \geq L_{AB}$

(6) $y_1 t_1 + y_2 t_2 + y_3 t_3 \geq L_{CD}$

(7) $z_1 t_1 + z_2 t_2 + z_3 t_3 \geq L_{EF}$

It is easy to imagine adding various terms to these inequalities to allow for time to switch modes, initial latencies, etc.

Note that the upper bounds are not used in this calculation. They are relevant to implementation requirements, because achieving a given lower rate bound on one channel may depend upon assuming an upper bound on the rate at which a competing channel is being used. Also, the upper bounds give a better description of the running system in a way that is quite relevant when questions arise concerning how one might add tasks. A nonlinear problem would arise if we did not take the t_i's to be constants, but variables meeting some extra constraints. This might be desirable if, for instance, we allow rate interpolation as mentioned above, or in other cases when a large class of feasible modes can be conveniently described parametrically.

One cannot expect to solve an instance of the schedulability problem as stated above unless one knows which message transmission modes are feasible. It is informative to consider the following precise mathematical problem, derived by stipulating a finite set of feasible modes and considering only PSM schedules that use only modes from that set. The *finitely constrained* schedulability problem (for Packet Stream Analysis) is then this: given a simple requirement set *Req* and a finite set **M** of message transmission modes, is there a PSM schedule that is adequate for *Req* and uses only modes from **M**?

The key idea of the following assertion is the same as in the example above:

Theorem 1 *There is an algorithm which decides the finitely constrained schedulability problem for Packet Stream Analysis.*

Proof. Suppose $Req = \{R_1, \ldots, R_q\}$ is a simple requirement set and $\mathbf{M} = \{M_1, \ldots, M_m\}$ is a finite set of modes. Enumerate in increasing order as $x_1 < x_2 < \ldots < x_p$ all numbers occurring as ready times or deadlines in *Req*. If a PSM schedule S is adequate for *Req* and $1 \leq i < p$, then one can rearrange and recombine the mode usage by S between x_i and x_{i+1} so that no mode is used more than once on this interval, while preserving the adequacy of the schedule. In fact, this schedulability problem is equivalent to the existence of a solution to the set of linear inequalities constructed using variables t_{ij} (for $1 \leq i < p$ and $1 \leq j \leq m$) to stand for the amount of time mode M_j is used between x_i and x_{i+1} and having inequalities of three classes: first, $t_{ij} \geq 0$, (for $1 \leq i < p$ and $1 \leq j \leq m$), second, for $1 \leq i < p$,

$$\sum_{j=1}^{m} t_{ij} \leq x_{i+1} - x_i,$$

and third, for each R in *Req*,

$$\sum_{i=a}^{b} \sum_{j=1}^{m} \alpha_j t_{ij} \geq L,$$

where L is the length of R, x_a is the ready time of R, x_b is the deadline of R, and α_j is the minimum rate guaranteed by M_j for the channel of R.
Q. E. D.

This proof is mathematically valid in any case, but the schedules it implicitly produces are reasonable only if one makes use of the assumption mentioned above about the possibility of padding.

3.4.6 Summary of Packet Stream Analysis

This approach does seem to provide an informative description of an important class of real problems. Especially interesting is the idea that it is helpful to think of network capacity in terms of feasible message transmission modes. Analogues of this notion might be relevant to other suggestions for analytical frameworks. Even if the only practical alternatives are the extremes of blasting and very low rate regulated transmission, this approach might still be useful in establishing guarantees for complex or somewhat dynamic data flow patterns.

The finitely constrained schedulability problems generated by a brute force application of Packet Stream Analysis may be quite useful for small systems or for restricted parts of larger systems (perhaps looking at one horizontal or vertical line). Larger problems may be mathematically tractable if approximated by linear programming problems using only a reasonable number of modes, but the validity of schedules generated as solutions to large systems of simultaneous constraints would be somewhat suspect. They might be hard to understand, and hence hard to work with or even to trust. Certainly the robustness of solutions in the face of inaccuracy of the input parameters would need to be carefully analyzed. Some structured way of applying the theory (perhaps hierarchically) is desirable if one is to be able to analyze potential system modifications easily. If one does not use it as a generator of schedules, but as an analytical framework, then Packet Stream Analysis might provide theoretical justification to more immediately intelligible scheduling techniques.

3.5 DISCRETE DEMAND

Here we present a notion of "discrete demand" that can be used to state resource requirements. The discrete demand will generally be slightly higher than the ordinary "utilization" for the same set of periodic requirements, but it has several potential advantages, especially when hierarchical approaches to subdividing the use of a communications network are to be considered. In particular, a justification of orthogonal mode scheduling requires a connection of an analysis of the lines to an analysis of the whole backplane. This is a second example of the general "pragmatic" strategy mentioned in Section 1.2. Whereas the development above of packet stream analysis involves a new suggestion for how to measure the capabilities of a communications network, discrete demand is proposed here as a more appropriate way to measure the requirements of applications.

In this subsection we first review an ordinary definition of utilization, summarize its basic role in giving schedulability guarantees, and point out some intrinsic drawbacks to its use for our purposes. Next we present the definition of discrete demand, which requires only a small modification of the definition of utilization. The discrete demand is stated in terms of a parameter (called Δ_t) that stands for the length of a relatively short interval of time. We then derive an estimate in terms of Δ_t for how much bigger the discrete demand can be than the ordinary utilization. Finally, we discuss how the notion of discrete demand alleviates some of the drawbacks of the ordinary notion of utilization in our context.

3.5.1 Utilization

We recall for comparison a common version of the definition of utilization. This definition is given for one arbitrary resource and any number of periodic tasks. In particular, it applies to the periodic message tasks mentioned in the discussion of the network subdivision approach (Section 3.2). A *periodic task* T is modelled by a period P, a ready time R, a length L and a deadline D. The length represents the amount of time required to complete the task, assuming the task has complete use of the resource. We also assume that

$$R < R + L \leq D,$$

and that $D \leq P$ (although this may fail in pipelined systems). For each iteration of T, call the interval from the task's release time to its deadline a *liveness interval* of T. Each liveness interval of T will have length $D - R$. The *utilization* $U(T)$ is defined by

$$U(T) = L/(D - R).$$

The utilization of a set of periodic tasks is just the sum of their individual utilizations.

Much work in the field of single resource scheduling theory is devoted to establishing that low utilization is a sufficient condition for a task set to be schedulable, often with extra assumptions about the task set and/or special nice properties guaranteed for an adequate derived schedule. A typical case would be when the tasks are assumed to be completely interruptible and the schedule is required to be based upon fixed priorities. See the paper of Malcolm and Zhao (Malcolm, 1994) for a relevant survey.

These sufficient bounds are not usually necessary. Cases where a task set with utilization greater than 1 is schedulable are usually neglected. There are two

different reasons why cases of utilization greater than 1 are of interest to us. Firstly, the treatment of the entire backplane as a single resource means that simultaneous heavy use of two completely disjoint channels can't be considered by the theory. One might try to introduce some location information into the notion of utilization, and we do not reject this approach entirely. In fact the notion of feasible patterns is a step in that direction. But we are looking here for a simpler theory, especially because we may actually treat subnetworks (like a line) as single resources for certain time intervals.

Consider secondly a task set which contains several tasks for which $D - R$ is very small, but whose utilization is only slightly less than 1. Such a task set might be easy to schedule, even though it has utilization much greater than 1, if the intervals from release to deadline of the various tasks in it are all well separated. This is significant for applications of interest, because the relative placement of message requirements within each overall processing period is pretty well known in advance.

Another obstacle to the use of this definition of utilization is that it works less well when the periods of the various tasks are not commensurate or have a large least common multiple. Finally, there are some difficulties integrating periodic hard constraints with aperiodic tasks and periodic tasks with soft constraints. For instance, some soft real-time constraints are better thought of as bandwidth requirements than multiple deadlines.

3.5.2 Simple Discrete Demand

It is convenient to think in terms of the following kind of contract between a scheduler and a task. There are agreed upon a fixed length of time Δ_t and a fraction d, with $0 \leq d \leq 1$. For convenience, let us say that an *aligned time quantum* means an interval of time from $n\Delta_t$ to $(n+1)\Delta_t$, for any nonnegative integer n. The scheduler agrees that the task will be given an opportunity to use the resource for at least $d\Delta_t$ units of time on each and every aligned time quantum. It is not assumed that the distribution of time actually allotted by the scheduler is periodic or regular, except that it should be distributed smoothly enough to be effectively usable. That is, it is assumed that the task can, on each iteration, actually use any time allotted it after its release time for that iteration and up to such time as it has completed that iteration.

The *(simple) discrete demand* $Q(T,\ \Delta_t)$ of a periodic task T with time quantum Δ_t is the least d for which such a contract suffices to guarantee that every deadline

of the repeated instances of T will be met. The discrete demand of a task set is the sum of the discrete demands of its elements. Since we are assuming $R+L \leq D$, it should be clear that $Q(T, \Delta_t)$ is well defined and has a value somewhere between 0 and 1.

We now estimate the discrete demand in a way which allows it to be compared to the ordinary utilization. The larger values for discrete demand result essentially from paying a price "up front" for possible edge effects. Assume that $3\Delta_t \leq (D-R)$. In this case, each liveness interval of T will contain an interval which is a union of consecutive aligned time quanta and has length at least $(D-R-2\Delta_t)$, no matter how the release time and deadline are positioned with respect to the multiples of Δ_t. This means that any fraction d suffices in the contract, as long as

$$d \times (D-R-2\Delta_t) \geq L.$$

This means that

$$Q(T, \Delta_t) \leq L/(D-R-2\Delta_t).$$

Thus, discrete demand gives a pretty good approximation to ordinary utilization as long as Δ_t is small compared to the least $(D-R)$ of the task set. Note that we are not assuming that P is an integral multiple of Δ_t for this particular calculation.

Furthermore, for aperiodic tasks or tasks with soft real-time deadlines, it would be reasonable in designing systems to require that their requirements be expressed in terms of discrete demand. Discrete demand gives, in fact, a strong version of a bandwidth guarantee, because it establishes particular points of time at which bandwidth guarantees will be met.

We have not addressed issues of task switching time or synchronization in distributed systems, but it seems that if a task set has total discrete demand considerably less than 1, then a scheduler should be able to satisfy its demands and still maintain considerable run-time freedom about exactly how it doles out resource time. It just needs to make sure that certain commitments are met at each multiple of Δ_t.

3.5.3 Time-dependent Discrete Demand

In order to consider discrete demand in more detail, we add another argument, standing intuitively for which aligned time interval is current. This allows a better assignment of combined demand to a task set, even while still assuming the resource to be indivisible. If T is a periodic task, and I is any aligned time

quantum (for Δ_t), then we define $Q^*(T, I, \Delta_t)$, the *(time dependent) discrete demand* of T on interval I with time quantum Δ_t rather brutally:

(a) If I intersects any liveness interval of T, then

$$Q^*(T, I, \Delta_t) = Q(T, \Delta_t).$$

(b) Otherwise,

$$Q^*(T, I, \Delta_t) = 0.$$

If Req is a set of periodic tasks, then we define

$$Q^*_\Sigma(Req, I, \Delta_t) = \sum_{T \in Req} Q^*(T, I, \Delta_t).$$

Finally, $D_Q(Req, \Delta_t)$, the *(total) quantized demand* of Req with time quantum Δ_t, means the maximum of the $Q^*_\Sigma(Req, I)$ for all aligned time quanta I.

Informally, to say that a set of periodic tasks has total quantized demand less than 1 is to say that for all aligned time quanta the total demand of all of the tasks is manageable, and thus it should be schedulable.

If the no-corners assumption is met, as is assumed for orthogonal mode scheduling, then one can divide up the set of periodic tasks into a horizontal and a vertical subset. If, for every aligned time interval I, either the horizontal or the vertical subset has zero discrete demand, while the other has less than 1, then it is plausible that OMS can work. One may not actually want to switch modes as fast as Δ_t, but one may not need to either.

3.5.4 The Choice of the Time Quantum

We summarize some trade-offs concerning the size of Δ_t.

- Larger Δ_t allow less overhead between aligned time quanta.
- Larger Δ_t allow more flexibility and efficiency for scheduling algorithms within the aligned time quanta.
- Smaller Δ_t mean that Q is closer to U.
- Smaller Δ_t may mean that smaller message buffers are needed.

3.6 INTERFACES AND INTEGRATION

From the point of view of an applications programmer using a mature real-time communications scheduling facility, there would be additional library calls available that are like a more elaborate version of the extra calls available to a programmer using a real-time operating system. These calls give both local and global directions to what the programmer might think of as an abstract entity called the communications scheduler. Local information controls the communication of data at certain points in the program. Some information about priority, delivery deadline, what to do in emergencies, etc. would be included over and above the stipulations necessary for an ordinary message based communications paradigm.

The global directions amount to telling the communications scheduler to make certain assumptions about the overall message flow pattern. In the near term we expect the required information to be a product of an augmented partitioning and mapping activity, including a detailed algorithm data flow analysis. Preliminary analysis of feasibility would be required, which might be automated to various degrees. It seems clear that not all details can be hidden from the applications programmers using such a communications scheduling facility, even if higher level languages and improved MPP compiler technology might eventually alleviate the extra burden considerably.

In making our network subdivision scheduling proposals progressively more specific, we have already had to begin considering where the entities involved in providing services would be located and how they would interact at run time. These details will always be highly dependent upon the exact machine architecture and choice of operating system, etc. As a last resort (but probably first implemented!) services can be integrated into the applications processes themselves. For uniformity and ease of use it is more desirable to have a server in the style of Mach for many services. But more efficiency could be gained by modifying the operating system and message passing software. Eventually some low level services, for instance, assuring a stipulated rate of transmission of a stream of packets, might be assisted by sophisticated router nodes or other hardware level mechanisms.

Finally, as was touched on above, the communications scheduling component interacts with the other components of high performance computing at the quite different level of how performance requirements and guarantees are to be given. For instance, the Paragon developers might state that at most ten percent of the backplane message traffic is overhead from system control, synchronization

messages, and the like. This is not good enough for us, because that reasonable overhead fraction might be unfavorably distributed. Similarly, it is not good enough for a real-time operating system to guarantee that certain messages will be prepared and ready to be sent by certain times, because unexpectedly early transmission of certain messages could interfere with earlier deadlines. On the other hand, the kinds of guarantees that we require are not unreasonable. One of our current concerns is how to devise a set of performance questions about the Paragon that we can reasonably expect to get hard answers to and that are sufficient to allow us to devise a communications scheduling scheme with guaranteed performance.

SECTION 4

CONCLUSION

To successfully apply MPPs to real-time embedded applications, four essential components must be in place. Efficient processing and message passing will be needed to meet the size, weight, and power constraints of embedded applications. System software support to schedule processing and communications will be needed to meet the real-time requirements. The communications scheduling component is presently missing from the real-time MPP research agenda. Commercial MPPs today have a tremendous amount of raw network capacity and the tendency is to assume that such high capacity translates into predictable communications performance. This is a mistake as our experiences have shown, and it will become even less the case for message passing techniques that provide higher data rates than current commercial techniques.

This paper has framed the communication scheduling problem and presented some of our initial work towards its solution. A variety of minor topics, especially relating to scheduling of a single line of the backplane, have not been included. We have described a general hierarchical scheduling approach called network subdivision and a more specific idea relevant to certain applications called orthogonal mode scheduling. We have also suggested an analytical framework called packet stream analysis and a modification of the ordinary notion of utilization called discrete demand.

These suggestions should be understood in terms of our overall goal to enhance the current state of the art to provide a combination of

1. performance parameters on hardware and support software that are really measurable or predictable with a high degree of confidence,

2. requirements parameters that adequately express the communications needs of target applications, and

3. scheduling algorithms, protocols, and mathematical analysis that justify a translation of applications requirements into performance parameters that we can guarantee are sufficient.

Packet stream analysis is not yet a mature analytic framework, but it does seem to present a valid and informative treatment of a significant class of cases. Its applicability depends greatly upon whether the level of granularity it assumes is appropriate and how good a facility for regulated sending is available. It is desirable now to determine robust and useful message transmission modes and to integrate this mathematical model with some form of hierarchical scheduling methodology in the style of the network subdivision approach.

The notion of discrete demand, especially in its time dependent form, may be a convenient way to justify versions of network subdivision, including the relatively intact transfer of some established scheduling theory. It also provides a more uniform treatment of hard and soft real-time requirements and helps get around problems regarding incompatible periods. There is a granularity issue here too: in some cases it may be difficult to find an acceptable value for the time quantum. Using time dependent discrete demand will require more detailed specification of the system than is thought desirable, for example, for some applications to uniprocessor scheduling of analysis based upon ordinary utilization. But achieving sufficiently efficient use of the distributed resources of scalable MPP's may intrinsically require comparatively detailed specification and complex analysis.

This is a relatively new area in terms of relevant published literature, and theoretical work could be profitably pursued in many directions. But we feel that it is important to let the near-term course of theoretical development be strongly guided by the needs of producing realistic scalable MPP demonstrations, probably of signal processing applications. We are optimistic that our work so far can be extended and integrated to provide an adequate solution to the MPP communications scheduling problem for such demonstrations. The outstanding problem in this regard is the difficulty of obtaining robust and sufficiently informative measures of the communications capacity of relevant combinations of hardware and underlying software.

The longer term problem of providing a genuine communications scheduling service for scalable MPP's that enables complex systems to be designed, maintained and updated relatively easily is much more difficult. In fact, the tempting analogy of scheduling tasks on a single processor may be misleading in that the intrinsic complexity of massively parallel programs and machines may never allow as good a solution as one might expect can be achieved. It is nevertheless important to make as much progress as possible towards such a service. Otherwise communications scheduling may become a critical impediment to the use of general purpose hardware and software for demanding real-time applications.

LIST OF REFERENCES

Aras, C. M., J. F. Kurose, D. S. Reeves, and H. Schulzrinne, January 1994, "Real-Time Communication in Packet-Switched Networks," *IEEE Proceedings*, Vol. 82, pp. 122–139.

Brown, C. P., M. I. Flanzbaum, R. A. Games, and J. D. Ramsdell, 1994, *Real-Time Embedded High Performance Computing: Application Benchmarks*, MTR 94B145, The MITRE Corporation, Bedford, Massachusetts.

Games, R. A. and D. S. Pyrik, 1994, *Parallel Implementation of the Planar Subarray Processing Algorithm*, MTR 94B114, The MITRE Corporation, Bedford, Massachusetts.

Linderman, R. W. and R. L. Kohler, Jr., November 1992, "Rapid Real-Time Signal Processing Systems Prototyping," *GOMAC 1992 Digest of Papers.*

Liu, C. L. and J. W. Layland, January 1973, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *JACM*, Vol. 20, pp. 46–61.

Malcolm, N. and W. Zhao, 1994, *Hard Real-Time Communication with Local Area Networks*, preprint, Department of Computer Science, Texas A&M University, College Station, Texas 77843–3112.

Ramamritham, K. and J. A. Stankovic, January 1994, "Scheduling Algorithms and Operating Systems Support for Real-Time Systems," *IEEE Proceedings*, Vol. 82, pp. 55–67.

Sha, L. and S. S. Sathaye, September 1993, "A Systematic Approach to Designing Distributed Real-Time Systems," *IEEE Computer*, Vol. 26, pp. 68–78.

von Eicken, T., D. E. Culler, S. C. Goldstein, and K. E. Schauser, March 1992, *Active Messages: A Mechanism for Integrated Communication and Computation*, UCB/CSD 92/#675, Computer Science Division — EECS, University of California, Berkeley.

Wheat, S. R., R. Riesen, A. B. Maccabe, D. W. van Dresser, and T. M. Stallcup, 1993, *PUMA: An Operating System for Massively Parallel Systems*, preprint, Sandia National Laboratories.

Zhao, W., J. A. Stankovic, and K. Ramamritham, September 1990, "A Window Protocol for Transmission of Time-Constrained Messages," *IEEE Tran. Computers*, Vol. 39, pp. 1186–1203.

DISTRIBUTION LIST

addresses	number of copies
PAUL F. SIERAK ROME LABORATORY/C3CB 525 BROOKS ROAD GRIFFISS AFB NY 13441-4505	20
MITRE CORPORATION MAIL STOP E025 202 BURLINGTON ROAD BEDFORD MA 01730	2
RL/SUL TECHNICAL LIBRARY 26 ELECTRONIC PKY GRIFFISS AFB NY 13441-4514	1
ADMINISTRATOR DEFENSE TECHNICAL INFO CENTER DTIC-FDAC CAMERON STATION BUILDING 5 ALEXANDRIA VA 22304-6145	2
ADVANCED RESEARCH PROJECTS AGENCY 3701 NORTH FAIRFAX DRIVE ARLINGTON VA 22203-1714	1
NAVAL WARFARE ASSESSMENT CENTER GIDEP OPERATIONS CENTER/CODE QA-50 ATTN: E RICHARDS CORONA CA 91718-5000	1
WRIGHT LABORATORY/AAAI-2 ATTN: MR FRANKLIN HUTSON WRIGHT-PATTERSON AFB OH 45433-6543	1
AFIT/LDEE 2950 P STREET WRIGHT-PATTERSON AFB OH 45433-6577	1

WRIGHT LABORATORY/MTEL
WRIGHT-PATTERSON AFB OH 45433 1

AAMRL/HE
WRIGHT-PATTERSON AFB OH 45433-6573 1

AUL/LSE
BLDG 1405
MAXWELL AFB AL 36112-5564 1

US ARMY STRATEGIC DEF
CSSD-IM-PA
PO BOX 1500
HUNTSVILLE AL 35807-3801 1

COMMANDING OFFICER
NAVAL AVIONICS CENTER
LIBRARY D/765
INDIANAPOLIS IN 46219-2189 1

COMMANDING OFFICER
NCCOSC RDTE DIVISION
CODE 02748, TECH LIBRARY
53560 HULL STREET
SAN DIEGO CA 92152-5001 1

CMDR
NAVAL WEAPONS CENTER
TECHNICAL LIBRARY/C3431
CHINA LAKE CA 93555-6001 1

SPACE & NAVAL WARFARE SYSTEMS COMM
WASHINGTON DC 20363-5100 1

CDR, U.S. ARMY MISSILE COMMAND
REDSTONE SCIENTIFIC INFO CENTER
AMSMI-RD-CS-R/ILL DOCUMENTS
REDSTONE ARSENAL AL 35898-5241 2

ADVISORY GROUP ON ELECTRON DEVICES 2
ATTN: DOCUMENTS
2011 CRYSTAL DRIVE,SUITE 307
ARLINGTON VA 22202

REPORT COLLECTION, RESEARCH LIBRARY 1
MS P364
LOS ALAMOS NATIONAL LABORATORY
LOS ALAMOS NM 87545

AEDC LIBRARY 1
TECH FILES/MS-100
ARNOLD AFB TN 37389

COMMANDER/USAISC 1
ATTN: ASOP-DO-TL
BLDG 61801
FT HUACHUCA AZ 85613-5000

AIR WEATHER SERVICE TECHNICAL LIB 1
FL 4414
SCOTT AFB IL 62225-5458

AFIWC/MSO 1
102 HALL BLVD STE 315
SAN ANTONIO TX 78243-7016

SOFTWARE ENGINEERING INST (SEI) 1
TECHNICAL LIBRARY
5000 FORBES AVE
PITTSBURGH PA 15213

DIRECTOR NSA/CSS 1
W157
9800 SAVAGE ROAD
FORT MEADE MD 21055-6000

NSA 1
ATTN: D. ALLEY
DIV X911
9800 SAVAGE ROAD
FT MEADE MD 20755-6000

DOD 1
R31
9800 SAVAGE ROAD
FT. MEADE MD 20755-6000

DIRNSA 1
R509
9800 SAVAGE ROAD
FT MEADE MD 20775

ESC/IC 1
50 GRIFFISS STREET
HANSCOM AFB MA 01731-1619

FL 2807/RESEARCH LIBRARY 1
OL AA/SULL
HANSCOM AFB MA 01731-5000

TECHNICAL REPORTS CENTER 1
MAIL DROP D130
BURLINGTON ROAD
BEDFORD MA 01731

DEFENSE TECHNOLOGY SEC ADMIN (DTSA) 1
ATTN: STTD/PATRICK SULLIVAN
400 ARMY NAVY DRIVE
SUITE 300
ARLINGTON VA 22202

Rome Laboratory

Customer Satisfaction Survey

RL-TR-_____

Please complete this survey, and mail to RL/IMPS, 26 Electronic Pky, Griffiss AFB NY 13441-4514. Your assessment and feedback regarding this technical report will allow Rome Laboratory to have a vehicle to continuously improve our methods of research, publication, and customer satisfaction. Your assistance is greatly appreciated.
Thank You

_____

_____

Organization Name:_____(Optional)

Organization POC: _____(Optional)

Address:_____

1. On a scale of 1 to 5 how would you rate the technology developed under this research?

5-Extremely Useful 1-Not Useful/Wasteful

Rating_____

Please use the space below to comment on your rating. Please suggest improvements. Use the back of this sheet if necessary.

2. Do any specific areas of the report stand out as exceptional?

Yes___ No_____

If yes, please identify the area(s), and comment on what aspects make them "stand out."

3. Do any specific areas of the report stand out as inferior?

Yes___ No___

If yes, please identify the area(s), and comment on what aspects make them "stand out."

4. Please utilize the space below to comment on any other aspects of the report. Comments on both technical content and reporting format are desired.

☆U.S. GOVERNMENT PRINTING OFFICE: 1995-710-126-20056

MISSION

OF

ROME LABORATORY

Mission. The mission of Rome Laboratory is to advance the science and technologies of command, control, communications and intelligence and to transition them into systems to meet customer needs. To achieve this, Rome Lab:

a. Conducts vigorous research, development and test programs in all applicable technologies;

b. Transitions technology to current and future systems to improve operational capability, readiness, and supportability;

c. Provides a full range of technical support to Air Force Materiel Command product centers and other Air Force organizations;

d. Promotes transfer of technology to the private sector;

e. Maintains leading edge technological expertise in the areas of surveillance, communications, command and control, intelligence, reliability science, electro-magnetic technology, photonics, signal processing, and computational science.

The thrust areas of technical competence include: Surveillance, Communications, Command and Control, Intelligence, Signal Processing, Computer Science and Technology, Electromagnetic Technology, Photonics and Reliability Sciences.